Dugong: Composing Uncertified DAG Consensus

George Danezis[1,2], Lefteris Kokoris-Kogias[1], Alberto Sonnino[1,2]
[1]*Mysten Labs* [2]*University College London (UCL)*

***Abstract*—There is a significant number of uncertified DAG consensus protocols spanning partial synchrony and asynchrony as well as crash, Byzantine, and hybrid fault models, yet each protocol is designed, proved, and implemented from scratch. We introduce Dugong, a single parameterized protocol that unifies uncertified DAG consensus protocols. Its core algorithm, the Decider can implement eight existing protocols and yields four new ones by just tuning its parameters. Given the property that DAGs decouple throughput from latency Dugong presents a simple trade-off between latency and resilience to network and node faults. An operator can adapt the parameters of their protocol epoch-by-epoch to adjust the resilience without needing to develop a new protocol or worry about throughput. Around the Decider, Dugong layers production feature mechanisms (adaptive leader scheduling, checkpoints, responsive liveness, and crash recovery) that provably compose with each other and with the Decider's configurations. We formalize Dugong in Lean 4, proving safety and liveness under the established definitions of Byzantine atomic broadcast, and identify and repair gaps in published proofs. Finally, we implement Dugong as a reusable Rust library paired with a discrete-event network simulator, enabling direct latency comparisons across network and fault models.**

1. Introduction

State machine replication (SMR) is the foundation of fault-tolerant distributed systems and blockchains [1], [2]. Over the last decade, consensus protocols built over a directed acyclic graph (DAG) of blocks have become an established architecture for high-throughput replication (Section 8). Among them, uncertified DAG protocols [3]–[6] have been deployed in Hedera, Sui, and IOTA because of their low latency. In DAG protocols every validator proposes a block of transactions in every round and references a subset of blocks from the preceding round. This structure separates data dissemination from transaction ordering [7] leading to a sepration of concerns. For the bulk data-dissemination validators batch and broadcast transactions concurrently, while for the delicate agreement protocol causal references of the DAG encode the votes. Once the DAG is constructed, validators totally order all blocks by deterministically interpreting their local view of the graph without sending additional consensus messages [8], [9].

In uncertified DAG protocols [5], [6] validators broadcast signed blocks without certifying them and encode votes and certificates in the DAG edges. Recent works have adapted uncertified DAGs to a variety of network models and fault thresholds. Under partial synchrony [2], Cordial Miners [5] and Mysticeti [6] tolerate f Byzantine faults among $n \geq 3f + 1$ validators and commit in three rounds; BlueBottle [10] commits in two rounds with $n \geq 5f + 1$; Nemo-Nemo [11] commits in two rounds under c crash faults with $n \geq 2c + 1$; and Orcaella [12] generalizes two-round commitment to the hybrid model with f Byzantine and c crash faults at $n \geq 5f + 3c + 1$. Under asynchrony, Cordial Miners [5], Mahi-Mahi [13], and asynchronous BlueBottle [10] replace leader timeouts with buffer rounds and a threshold common coin [14]. Although these protocols build on the same underlying DAG structure, each work starts largely from scratch, designing custom commit rules, writing standalone proofs and prototypes, and evaluating against only one or two baselines. Starting from scratch is error-prone, as safety violations in Black Marlin [15] and S-Minnow [16] show (Appendix C). Moreover, production deployments such as Sui and IOTA [17] require additional surrounding protocols, including leader reputation and scheduling [18], [19], checkpoints [12], [17], and crash recovery, whose interaction with different consensus rules has not been analyzed.

This paper takes an integrated approach to designing and analyzing uncertified DAG consensus. Instead of starting from a specific network assumption and fault model, we identify the communication patterns shared across uncertified DAG protocols and build Dugong (Section 3). At the core of Dugong is a single Decider algorithm that implements a parameterized decision rule on the DAG and unifies the ordering logic of this protocol family. By setting six rule parameters, a leader schedule, and a leader election, the Decider recovers eight existing protocols across partial synchrony, asynchrony, and crash, Byzantine, and hybrid fault models [5], [6], [10]–[13], and yields four new protocols: Asynchronous Orcaella, Asynchronous Nemo-Nemo, Hybrid Mysticeti ($n \geq 3f + 2c + 1$ under partial synchrony), and Hybrid Mahi-Mahi ($n \geq 3f + 2c + 1$ under asynchrony). Across all twelve configurations, the Decider captures the trade-off between resilience and latency at the same throughput: **tolerating a stronger adversary requires more rounds of communication to reach agreement**.

Around the Decider, Dugong integrates mechanisms needed in production deployments (Section 4). These include (a) two-tier finality checkpoints [12] above the consensus decisions, (b) leader scheduling [19], [20] alongside the Decider, (c) crash recovery, and (d) responsive liveness under partial synchrony. Since every protocol parameterizes

the same algorithm we can prove each composition once and compose them with one another.

We formalize Dugong and prove its safety, liveness, and composition with the mechanisms in the Lean 4 theorem prover (Section 5). We use the definitions of atomic broadcast from Sailfish [21] and Sui et al. [22]. Both existing and new configurations inherit these machine-checked guarantees. We also formalize each existing protocol on its own and check that it agrees with its Decider configuration. This formalization identified gaps in published proofs, including an omitted tie-breaking rule for BlueBottle's two-round indirect commits [10], [23], which our specification fixes.

Finally, we implement Dugong in Rust as a unified library supporting all existing protocols and new configurations. We develop a deterministic discrete-event network simulator (Section 6) and tune it with real inter-region latencies from AWS servers. Running the same implementation across all twelve configurations allows us to evaluate latency and fault tolerance on equal terms (Section 7) without the cost of large-scale cloud deployments.

Contributions. We make the following contributions:

- We present Dugong and its parameterized Decider, unifying uncertified DAG consensus to recover eight existing protocols and derive four new protocol configurations (Section 3).
- We compose production mechanisms around Dugong, including adaptive leader scheduling, checkpoints, responsive liveness, and crash recovery (Section 4).
- We formalize Dugong in Lean 4 [24], proving, under established definitions, the safety and liveness of the Decider configurations and their composition with the feature mechanisms, while identifying and fixing gaps in published proofs (Section 5).
- We implement Dugong in Rust [25] and evaluate all twelve configurations in a discrete-event simulator, quantifying the latency and fault-tolerance trade-offs of uncertified DAG consensus on equal terms (Sections 6 and 7).

2. Background and DAG-Based Consensus

DAG construction and local interpretation. DAG-based consensus protocols implement atomic broadcast by decoupling transaction dissemination from ordering [7]. A committee of n validators (or n units of stake) constructs a round-based directed acyclic graph of blocks. In each round r, every correct validator proposes at most one signed block containing a batch of transactions and cryptographic hash references to parent blocks from round $r-1$. A non-genesis block is syntactically valid if it references at least a quorum of q blocks from round $r-1$ authored by distinct validators, includes its own author's round-$(r-1)$ block, and references at most one round-$(r-1)$ block per author. Besides its parents, every block carries *weak references*: a correct validator references every earlier-round block it holds that its new block would not otherwise reach. Weak references count toward no quorum, and they ensure that every correct block eventually enters the history of every later correct block, even one that no later block takes as a parent [8], [21].

A block b' is in the *causal history* of b if b' is reachable from b by following parent references. Each validator maintains a local *view* of the DAG, which is a causally closed subset of valid blocks. A validator adds a block to its view only after downloading and validating the block's entire causal history with the help of a block synchronizer. Once blocks are in the local view, ordering requires no extra communication. Each validator applies a deterministic decision rule to its view to commit a sequence of *slots* (see below). Committing a leader block commits its entire causal history, which validators flatten in a deterministic order to produce the final transaction log [8].

Uncertified DAGs. In uncertified DAG protocols, validators broadcast signed blocks directly, in a single message delay per round. Because blocks are not certified prior to dissemination, a Byzantine validator can *equivocate* by sending different blocks for the same round to different validators. Uncertified DAG protocols decide *slots* and tolerate equivocation by encoding votes, blames, and certificates implicitly in the parent references of subsequent rounds:

- **Decision slots:** A slot (r, ℓ) is a round r and a leader index ℓ, whose leader validator is set by a schedule or, under asynchrony, a common coin. Its *leader blocks* are the leader's round-r blocks, several if it equivocates. Over a *wave* of rounds, from proposal through vote to decision, each slot is committed (one leader block), skipped, or left undecided. Committed slots, in round order, determine the delivered sequence.
- **Votes and blames:** A block b at a round above a leader slot (r, ℓ) *votes* for a candidate block L of that slot if L is the first block for (r, ℓ) encountered during a deterministic depth-first traversal of b's parent references. Because the traversal order is fixed and a valid block references at most one parent per author, a correct validator votes for at most one block per slot even when the leader equivocates. If b does not reach any block for (r, ℓ), b *blames* the slot.
- **Certificates:** In three-round protocols such as Mysticeti [6], a block C two rounds above the leader's proposal *certifies* L if C's parents include votes for L from at least a certificate threshold of distinct authors. In two-round protocols such as BlueBottle [10], Orcaella [12], and Nemo-Nemo [11], votes act directly as certificates without a separate certification round.

Network models, fault models, and trade-offs. Uncertified DAG protocols operate under different failure and timing assumptions, trading fault tolerance against commit latency. In the hybrid fault model [12], an adversary can corrupt up to f *Byzantine* validators, which may equivocate or deviate arbitrarily from the protocol, and up to c *crashed* validators, which follow the protocol until they permanently stop sending messages. The remaining $n-f-c$ validators are *correct*. Setting $c = 0$ yields the standard Byzantine fault model, while setting $f = 0$ yields the crash fault-tolerant (CFT) model. All quorum thresholds, except the skip threshold, are bounded by $n-f-c$ so that the system

can make progress even when all $f + c$ faulty validators are silent. As a special case, the skip threshold may be n, indicating no direct skip, leaving undecided slots to be decided by the indirect rule.

For network timing, protocols assume either partial synchrony [2] or full asynchrony [1]. Under partial synchrony, message delays between correct validators are bounded by Δ after an unknown Global Stabilization Time (GST). Partially synchronous protocols use a predefined leader schedule and wait up to a timeout for each round's leader blocks before advancing to the next round, ensuring deterministic liveness after GST. The schedule may also adapt to the committed history [19], [20], and a responsive validator advances as soon as it holds the blocks the rule counts (Section 4). Under asynchrony, message delays are unbounded. To circumvent the FLP impossibility result [1], validators in asynchronous protocols advance to the next round as soon as they receive a quorum of blocks from the current round, without waiting for specific leaders, and elect each slot's leader retroactively using a threshold common coin [14] revealed in the final round of the wave [5], [10], [13].

Problem definition. Each validator *delivers* a sequence of blocks. A protocol implements *atomic broadcast* [8], [21], [22] against up to f Byzantine and c crashed validators when it satisfies four properties. *Agreement*: if a correct validator delivers a block b, every correct validator eventually delivers b. *Integrity*: every correct validator delivers each block at most once, and delivers a block of a correct validator only if that validator built it. *Validity*: every block built by a correct validator is eventually delivered by every correct validator. *Total order*: if two correct validators deliver blocks b and b', they deliver them in the same order. It is *Byzantine* atomic broadcast when $f > 0$, and crash-tolerant when $f = 0$. We also prove a stronger safety property, *prefix consistency*: the delivered sequences of any two correct validators are prefixes of one another. Prefix consistency, Total order, and Integrity hold in every execution, whatever the message delays, before GST and under asynchrony alike. Agreement and Validity are liveness properties. Agreement needs only that messages between correct validators are eventually delivered. Validity holds after GST under partial synchrony, and with probability 1 under asynchrony against an adversary that controls message delays but cannot predict the common coin.

The fault and network assumptions determine the shortest wave length, namely the number of rounds needed for a direct commit. Under partial synchrony, tolerating $n \geq 3f + 1$ requires three rounds per wave (propose, vote, and certify) so that any two certificate quorums intersect in at least one correct validator, and extending this three-round rule to hybrid faults requires $n \geq 3f + 2c + 1$. Committing in two rounds eliminates the certificate round and therefore requires a larger committee to prevent an equivocating leader from splitting the vote quorum: $n \geq 5f + 1$ under Byzantine faults [10] and $n \geq 5f + 3c + 1$ under hybrid faults [12]. Under crash faults, no leader equivocates, and Nemo-Nemo [11] commits in two rounds with $n \geq 2c + 1$ by giving up direct skips (Section 3.4). Adapting any of these protocols to asynchrony preserves its committee size and quorum thresholds while inserting one or two *buffer rounds* between the propose and vote rounds, so that proposals spread before the common coin reveals the leader (Section 3.4).

Existing uncertified DAG protocols and common message patterns. While Cordial Miners [5], Mysticeti [6], Mahi-Mahi [13], BlueBottle [10], Orcaella [12], and Nemo-Nemo [11] target different network and fault assumptions, their commit rules all check the same two graph patterns over each wave:

- **Direct decision rule:** At the decision round of a slot's wave, a validator examines its local view. If at least a commit threshold of distinct authors in the decision round certify a candidate block L, the validator directly commits L. Conversely, if at least a skip threshold of distinct authors in the vote round blame the slot, the validator directly skips the slot.
- **Indirect decision rule:** If neither threshold is met, the slot remains undecided at the end of its wave. The validator then decides the slot indirectly from the earliest subsequent wave whose leader block A (the *anchor*) is committed: if A's causal history contains at least a link threshold of decision-round certificates for a candidate L (breaking ties by the least block identifier if multiple equivocating blocks qualify), the validator commits L, and otherwise it skips the slot.

Across all of these protocols, safety relies on three quorum-intersection properties between the direct and indirect rules: (i) a slot that is directly skipped in one view cannot have enough certificates in an anchor's causal history to be indirectly committed in another view; (ii) a directly committed block cannot be outvoted at an anchor by a conflicting block from the same slot; and (iii) a directly committed block is guaranteed to have enough certificates in the causal history of every subsequent valid anchor to be indirectly committed by any validator. In Section 3, we capture this common decision logic in a single parameterized Decider.

3. A Parameterized Decider

Dugong separates the parts that uncertified DAG protocols share from the parts that vary across configurations. Every configuration uses the same DAG construction and the same direct and indirect decision rules, while varying the wave shape, thresholds, parent quorum, leader schedule, and election mode. This section describes the common part, as a single Decider algorithm that takes a DAG and returns, for each slot, whether it is committed, skipped, or undecided. Unlike classical consensus protocols, the Decider is not event-driven: it is a deterministic function from a local view of the DAG to decisions, and leaves messages, timers, and synchronization to the DAG layer (Section 2).

Every configuration of Dugong runs on the DAG layer of Section 2. We call a block's reference to its author's previous block its *self-parent*, and the number of previous-round

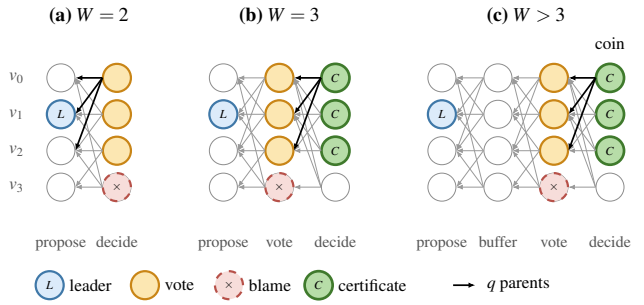


Figure 1. Wave shapes for $W = 2$, $W = 3$, and $W > 3$ ($q = 3$).

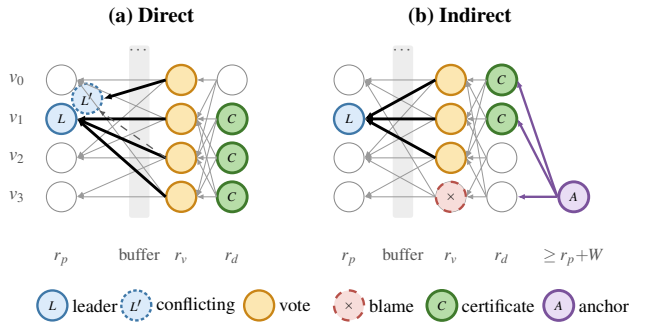


Figure 2. Direct (a) and indirect (b) rules for any wave length. In (a), the leader equivocates.

blocks every block references, and every validator waits for before advancing, the *parent quorum* q (Section 3.2).

As the DAG grows, the Decider decides the slots of Section 2, whose leaders the schedule assigns (Section 3.2). Each slot spans a wave of W consecutive rounds from its *proposal round* to its *decision round* (Figure 1). Validators cast votes in the *vote round*, which equals the decision round when votes act as certificates and precedes it by one round otherwise. Any rounds between the proposal round and the vote round are *buffer rounds*.

Challenges. Unifying these rules into a single Decider requires resolving two issues. First, direct and indirect decisions must agree. Protocols committing in three or more rounds [5], [6] have unique certificates, thus a single certificate in the anchor's history suffices for safety. Without certificates, however, conflicting blocks can collect votes, and deciding indirectly via the same rule is unsafe. This is not just of theoretical difficulty as both Black Marlin [15] and S-Minnow [16] suffered this issue(Appendix C). The Decider instead treats votes as certificates and sets *link* low enough for every anchor to reach after a direct commit, yet too high for a conflicting block to clear it. Second, existing commit rules diverge once buffer rounds or equivocating blocks appear. Counting blames right after the proposal round misses votes cast through buffer blocks. Testing skips per candidate block is unsafe: a validator that first tests an equivocating block with no votes skips the whole slot, while another validator commits the leader's other block. The Decider counts slot-level blames in the vote round and separates the parent quorum q from the safety thresholds.

3.1. The Decider

The Decider implements the direct and indirect rules of Section 2. Algorithm 1 gives the definitions it uses, the commit loop, and the two rules. Figure 2 illustrates both rules for an arbitrary wave length.

Commit loop. TRYDECIDE runs whenever the local view grows. It visits the slots the schedule defines (SLOTS), newest first, from the highest round r_{high} down to r_{last}, the round of the last decision the caller consumed (line 3), and decides each slot by the direct rule first and the indirect rule second (lines 4 and 5). The decisions accumulate in a list S,

Algorithm 1 The Decider (parameters in Section 3.2).

Notation
DAG[r]: round-r blocks in the local view
authors X: distinct authors of the blocks in X
Decisions: Commit(L), Skip, or $\perp$ (undecided)
Slots
SLOTS lists slots newest first, by round then leader index:
every round if *pipeline*, else every W-th round
$r_p(s) = s.round$ ▷ proposal round
$r_d(s) = r_p(s) + W - 1$ ▷ decision round
$r_v(s) = r_d(s)$ if $voteIsCert$, else $r_d(s) - 1$ ▷ vote round
L ranges over $s.leader$'s blocks in round $r_p(s)$ ▷ leader blocks
Voting
VOTE(b, s): the block of slot s that b's history reaches first by depth-first search, or $\perp$ if none (b blames s)
C certifies L: VOTE$(C, s) = L$ if $voteIsCert$, else C's parents include votes for L from at least $cert$ distinct authors
Counting
BLAMERS$(s) = \text{authors}\{b \in \text{DAG}[r_v(s)] : \text{VOTE}(b, s) = \perp\}$
CERTS$(s, L, H) = \text{authors}\{C \in H : C \text{ certifies } L\}$

1: **procedure** TRYDECIDE(r_{last}, r_{high})
2: $S \leftarrow [\,]$ ▷ decisions, oldest first
3: **for** $s \in$ SLOTS(r_{high} **down to** r_{last}) **do**
4: $d \leftarrow$ TRYDIRECT(s)
5: **if** $d = \perp$ **then** $d \leftarrow$ TRYINDIRECT(s, S)
6: $S \leftarrow d \parallel S$
7: **return** longest decided prefix of S

8: **procedure** TRYDIRECT(s)
9: **if** $|\text{BLAMERS}(s)| \geq$ *skip* **then** ▷ direct skip
10: **return** Skip
11: **return** Commit(L) for the least L with ▷ direct commit
$|\text{CERTS}(s, L, \text{DAG}[r_d(s)])| \geq$ *commit*, else $\perp$

12: **procedure** TRYINDIRECT(s, S)
13: $a \leftarrow$ first non-Skip decision in S at round $\geq r_p(s) + W$
14: **if** there is none, or $a = \perp$ **then return** $\perp$ ▷ no anchor yet
15: **let** Commit(A) $= a$ ▷ anchor A
16: **return** Commit(L) for the least L with ▷ indirect rule
$|\text{CERTS}(s, L, \text{DAG}[r_d(s)] \cap \text{history}(A))| \geq$ *link*, else Skip

oldest slot first, so that every slot above the current one has been examined when the indirect rule looks for an anchor. The loop returns the longest prefix of S in which every slot is decided (line 7), and the caller commits the sub-DAGs of its committed leaders in order. An undecided slot therefore blocks delivery of every slot above it, so an equivocating leader delays subsequent waves in a pipelined protocol

(Section 7.4). Restarting from the last consumed round is idempotent because revisited slots keep their decisions, and the caller drops them (Appendix A).

Votes and blames. A block b above a slot votes for one block of that slot in its causal history, chosen by a deterministic vote rule (VOTE). If its history holds no block of the slot, b blames the slot. Without equivocation the history holds at most one block of the slot, and every rule picks it. When the leader equivocates, b's history may contain several of the leader's blocks, and VOTE picks one, so b votes for at most one block per slot. Our proofs hold for any deterministic VOTE, and Dugong uses depth-first traversal in every configuration.

Direct rule. TRYDIRECT skips the slot on *skip* blames in the vote round (line 9), and otherwise commits a leader block L on $commit$ certificates in the decision round (line 11). A certificate is the vote itself when $voteIsCert$ holds, and a block referencing $cert$ votes for L otherwise (C certifies L in Algorithm 1). The thresholds ensure that at most one of an equivocating leader's conflicting blocks reaches $commit$, so taking the least identifier only makes the choice definite. A slot that reaches neither threshold stays undecided ($\perp$).

Indirect rule. TRYINDIRECT resolves a slot that the direct rule leaves undecided, using a committed leader of a later wave. It scans the decisions at least one wave above the slot, oldest first, and passes over skipped slots (line 13). If the first remaining decision is undecided, so is the slot. Otherwise that decision commits a block A, the *anchor*. A commits the leader block with at least $link$ certificates in its causal history (line 16), or skips the slot if no block qualifies. When $link$ is low enough that two conflicting blocks of an equivocating leader both qualify, the least identifier breaks the tie. Anchors sit at least a full wave above the slot so that every block there reaches at least q of the slot's decision-round blocks (Section 3.4).

3.2. Parameters

The Decider has six parameters, which determine how each slot is decided. The parameters are:

- W: rounds from proposal to decision. The shortest wave has two rounds, proposal and vote. A separate certificate round adds one, so $W \geq 3$ when votes are not certificates, and any further rounds are buffer rounds.
- $voteIsCert$: whether a vote is its own certificate. When set, the vote round is the decision round and $cert$ is unused.
- $commit$: distinct certifying authors in the decision round needed for a direct commit.
- $skip$: distinct blaming authors in the vote round needed for a direct skip. Setting $skip = n$ disables direct skips because on a DAG with self-parents the leader's own vote-round block never blames its slot.
- $cert$: distinct voting authors a decision-round block must reference to certify a leader (used when $voteIsCert$ is false).
- $link$: distinct certifying authors in the anchor's history needed for an indirect commit.

A protocol also fixes the schedule ($leaders$ and $pipeline$), the leader election ($async$), and the DAG's parent quorum q.

Schedule and election. $leaders$ is the number of leaders per round, and $pipeline$ controls whether a wave starts in every round, so that neighboring waves overlap, or only every W rounds. Both parameters specify which slots exist without changing the decision rule, so the proofs treat the schedule separately. Under partial synchrony, $async$ is false and validators follow a public deterministic schedule (such as round-robin, leader reputation [19], or an adaptive schedule, Section 4) with the leader timeout of Section 2. Under asynchrony, $async$ is true and validators reveal a common coin [14] in the decision round, using buffer rounds to spread proposals before the coin names the leader.

Parent quorum. The parent quorum q is a DAG parameter that lower-bounds how many decision-round blocks any later block reaches. In the published three-round protocols q equals $cert$, but the two differ in the hybrid three-round rows of Table 1, where $cert = \lceil (n + f + 1)/2 \rceil$ falls below $q = n - f - c$ once $n > 3f + 2c + 2$. We therefore list q as its own column, as the implementation does (Section 6), and set $q = n - f - c$ throughout.

Liveness. Liveness requires thresholds $commit, cert \leq n - f - c$ (Section 2). The skip threshold *skip* is exempt, since the indirect rule decides the slots a direct skip leaves open.

3.3. Recovering Existing Protocols

The upper half of Table 1 lists eight published protocols as configurations of the Decider. We take each row's thresholds and wave length from the protocol's paper, and set $voteIsCert$ when the protocol has no separate certificate round. Each row makes the same decisions as its protocol, except possibly where VOTE breaks ties among an equivocating leader's blocks. Without equivocation, the decisions are identical. We prove in Lean that each row equals our formalization of its protocol (Section 5).

Three-round partially synchronous rows. Mysticeti is the Decider at $W = 3$ with every threshold at $n - f$ and a link of one certificate. Two certificate quorums intersect in a correct validator, so a slot has at most one certified block and a single certificate in the anchor's history identifies it. Cordial Miners is the same row without the direct skip, and with one leader per wave and no pipelining. Its published rule skips only through an anchor, which $skip = n$ encodes directly on a DAG with self-parents.

Two-round rows. BlueBottle has no certificate round ($voteIsCert$), so its vote round is its decision round. At $n = 5f + 1$, a direct commit needs $n - f$ votes and an indirect commit needs $n - 3f$ votes in the anchor's history. Orcaella extends BlueBottle to f Byzantine and c crashed validators at $n = 5f + 3c + 1$. Its commit and parent quorums are $n - f - c$, its link is $n - 3f - 2c$, and its skip threshold

TABLE 1. CONFIGURATIONS OF THE DECIDER. *NO DIRECT SKIP.

Protocol	Ref.	n	Network	Wave	Vote is cert.	Commit	Skip	Cert.	Link	q
Cordial Miners	[5]	$3f+1$	PS	3	×	$n-f$	n^*	$n-f$	1	$n-f$
Cordial Miners Async	[5]	$3f+1$	Async	5	×	$n-f$	n^*	$n-f$	1	$n-f$
Mysticeti	[6]	$3f+1$	PS	3	×	$n-f$	$n-f$	$n-f$	1	$n-f$
Mahi-Mahi	[13]	$3f+1$	Async	4 or 5	×	$n-f$	$n-f$	$n-f$	1	$n-f$
BlueBottle	[10]	$5f+1$	PS	2	✓	$n-f$	$n-f$	–	$n-3f$	$n-f$
BlueBottle Async	[10]	$5f+1$	Async	3	✓	$n-f$	$n-f$	–	$n-3f$	$n-f$
Orcaella	[12]	$5f+3c+1$	PS	2	✓	$n-f-c$	$4f+2c+1$	–	$n-3f-2c$	$n-f-c$
Nemo-Nemo	[11]	$2c+1$	PS	2	✓	$n-c$	n^*	–	1	$n-c$
Orcaella Async	*new!*	$5f+3c+1$	Async	3	✓	$n-f-c$	$4f+2c+1$	–	$n-3f-2c$	$n-f-c$
Nemo-Nemo Async	*new!*	$2c+1$	Async	3	✓	$n-c$	n^*	–	1	$n-c$
Hybrid Mysticeti	*new!*	$3f+2c+1$	PS	3	×	$2f+c+1$	$\lfloor \frac{n+f+1}{2} \rfloor$	$\lceil \frac{n+f+1}{2} \rceil$	1	$n-f-c$
Hybrid Mahi-Mahi	*new!*	$3f+2c+1$	Async	4 or 5	×	$2f+c+1$	$\lfloor \frac{n+f+1}{2} \rfloor$	$\lceil \frac{n+f+1}{2} \rceil$	1	$n-f-c$

is $4f+2c+1$, which equals $n-f-c$ at this bound. Nemo-Nemo commits in two rounds under crash faults alone, at $n = 2c+1$, with commit and parent quorums of $c+1$, a link of one, and no direct skip. Its published blocks have no self-parent. Dugong adds one, so that $skip = n$ disables the direct skip, as in Cordial Miners.

Asynchronous rows. Mahi-Mahi, BlueBottle Async, and Cordial Miners Async use the thresholds of Mysticeti, BlueBottle, and Cordial Miners, respectively. They set $async$, so a common coin opened in the decision round elects the leader, and add buffer rounds before the vote. Mahi-Mahi uses one or two buffer rounds ($W \in \{4, 5\}$), BlueBottle Async one, and Cordial Miners Async two. With buffer rounds, a voter's history can contain several blocks of an equivocating leader, and different VOTE rules can pick different ones.

3.4. New Configurations

The lower half of Table 1 lists four new protocols. The safety inequalities below fix their thresholds, and the Decider's proofs apply unchanged. All four are safe, and the asynchronous ones decide every slot with probability 1 against an adversary that cannot predict the coin.

Safety inequalities. Fix n validators with at most f Byzantine, a parent quorum q, and a row of the table. Two facts underlie the inequalities. First, two sets of same-round blocks with more than $n+f$ authors in total share a block by a non-Byzantine author, which votes for at most one block of a slot. Second, a block in a later round reaches at least $a+q-n-f$ of the decision-round blocks of any a authors. Each wave shape yields three inequalities:

Two rounds (voteIsCert)	*Three rounds*
$skip + link \geq n+f+1$	$2\,cert \geq n+f+1$
$commit + link \geq n+f+1$	$skip + cert \geq n+f+1$
$link + n + f \leq commit + q$	$commit + q \geq n+f+1$

In both columns, one inequality keeps a direct skip consistent with a later indirect commit ($skip+link$ and $skip+cert$). Another prevents conflicting decisions on an equivocating leader's blocks. In two rounds it keeps a direct commit consistent with an indirect decision on a conflicting block, and in three rounds it makes certificates unique. The last ensures that every anchor reaches enough certificates of a directly committed leader. The three-round family also fixes $link = 1$. If a row meets its column's inequalities, any two views decide every slot the same way, under every vote rule (Appendix A). Every row of Table 1 meets them except the two Nemo-Nemo rows, which meet their crash-only weakening (below).

Solving for the thresholds. Add the liveness bound $commit, cert \leq n-f-c$ and set $q = n-f-c$. The two-round inequalities have a solution if and only if $n \geq 5f+3c+1$. At this bound they force Orcaella's thresholds, and at $c = 0$ BlueBottle's. The three-round inequalities have a solution if and only if $n \geq 3f+2c+1$. At this bound they force $commit = 2f+c+1$ and $cert = \lceil (n+f+1)/2 \rceil$. The smallest skip threshold that $skip + cert \geq n+f+1$ allows is $\lfloor (n+f+1)/2 \rfloor$, and the correct validators alone reach it. We call this configuration *Hybrid Mysticeti*. At $c = 0$ it is Mysticeti. The bounds differ because two-round rules count votes, which conflicting blocks of an equivocating leader can both collect, so $link$ must exclude a conflicting block yet stay reachable from every anchor. A certificate round makes certificates unique, so one certificate suffices, at the cost of a round. For $f = 0$, no validator equivocates, so a slot has at most one block and the inequality against conflicting blocks drops out. The remaining two-round inequalities admit $n \geq 3c+1$ with a direct skip ($skip \leq n-c$) and $n \geq 2c+1$ without one ($skip = n$). Nemo-Nemo is the second case, and omitting the direct skip reduces its committee from $3c+1$ to $2c+1$.

Asynchronous configurations. Setting $async$ and adding buffer rounds to Orcaella, Nemo-Nemo, and Hybrid Mysticeti gives *Orcaella Async*, *Nemo-Nemo Async*, and *Hybrid Mahi-Mahi*. The inequalities do not depend on W, so safety is unchanged. Liveness depends on how many leaders are *safe to commit* (*safe* for short) before the coin opens: every valid vote-round block reaches the leader's proposal, so if the coin names that leader, its slot commits whatever the

scheduler does next. The scheduler acts before the coin is revealed, so it cannot single out the leader the coin will name, only limit how many leaders are safe. Every asynchronous row has $2q \geq n+f+1$, which gives a common core after two rounds. One buffer round then guarantees at least one safe leader, and two guarantee at least $q-f$. Further buffer rounds do not help: an adversarial scheduler can hold the count at $q-f$. When $2q = n+f+1$, as at $n = 3f+1$, $3f+2c+1$, and $2c+1$, one buffer round guarantees at least one safe leader, so the coin names a safe leader with probability at least $1/n$. Mahi-Mahi, Cordial Miners Async, and Hybrid Mahi-Mahi therefore use up to two buffer rounds. Nemo-Nemo Async uses one buffer round and guarantees at least one safe leader, as do Mahi-Mahi and Hybrid Mahi-Mahi at $W = 4$. At $n \geq 5f+3c+1$, one buffer round guarantees at least $n/3$ safe leaders, so the coin names a safe leader with probability at least $1/3$. BlueBottle Async and Orcaella Async therefore use one buffer round. With more than $n-s$ leaders per round, where s is the number of safe leaders, at least one leader of the round is safe.

All four configurations are implemented in our library (Section 6) and evaluated alongside the published protocols (Section 7). Hybrid Mysticeti and Hybrid Mahi-Mahi tolerate f Byzantine and c crashed validators with $3f+2c+1$ validators, compared to Orcaella's $5f+3c+1$, at the cost of one more round. Orcaella Async and Nemo-Nemo Async bring the two-round hybrid and crash rules to asynchrony with one buffer round. The two differ in how quickly they decide: Orcaella Async keeps at least $n/3$ safe leaders, so the coin names one with probability at least $1/3$ per wave, whereas Nemo-Nemo Async guarantees at least one, a probability of at least $1/n$. Both decide every slot with probability 1 (Lemma 5), but against an adversarial scheduler Nemo-Nemo Async may need on the order of n waves, trading worst-case latency for one fewer buffer round.

3.5. Fault Tolerance Versus Latency

Table 1 shows the trade-off between fault tolerance and latency. Every row builds the same DAG, one block per validator per round with q parents each, so throughput is the same and rows differ only in latency, measured in rounds. Three choices set this latency. First, for a given wave length, a stronger adversary needs a larger committee. Two-round waves need $5f+3c+1$ validators and three-round waves $3f+2c+1$, so at a fixed committee the three-round rule tolerates more faults and the two-round rule commits one round sooner. Second, asynchrony adds buffer rounds so proposals spread before the coin names the leader, one at $5f+1$, $5f+3c+1$, and $2c+1$, and up to two at $3f+1$ and $3f+2c+1$. Third, omitting the direct skip, as Cordial Miners and Nemo-Nemo do, adds no latency without faults, but leaves every crashed leader to the indirect rule, one wave later. Without faults, commit latency is W rounds, plus the wait for a leader slot without pipelining (Section 7).

4. Layering Mechanisms Around the Decider

The Decider algorithm is the heart of our consensus protocol as it achieves the hardest and most critical part, the total ordering. Algorithm 1 is only 16 lines of pseudocode. Its simplicity, however, is not sufficient for the needs of a dynamic, heterogeneous environment such as a production blockchain. We take Sui as an example. Sui deploys additional mechanisms along Mysticeti: (a) responsive liveness, so that rounds advance at network speed rather than on timeouts, (b) a leader scheduling algorithm that adapts slot assignments to validators' varying performance, (c) a crash recovery mechanism for each validator, which is outside the classic BFT adversary, and (d) a checkpoint tier that exposes higher resilience guarantees to light clients at the cost of one extra message delay. Each of these mechanisms is as critical and as error-prone as the core consensus algorithm. In 3 819 commits to Sui's consensus, checkpoint, and execution code we count 96 bug fixes related to them. Taking a holistic approach, Dugong provably composes each mechanism with the Decider as well as with the other mechanisms.

How the mechanisms compose. Every mechanism reads the Decider only through two properties: decisions agree across views, and a correctly led slot commits once its wave is built and certified (Section 5). Composing a mechanism with any configuration, or with the other mechanisms, therefore needs no new argument about the Decider. All four mechanisms compose under partial synchrony, and all but responsive liveness under asynchrony.

Mechanism 1: Responsive liveness. A responsive validator builds its next block as soon as it holds what the Decider requires: a block reaching the leader's block or the votes certifying it. It waits out the timeout only as a fallback. Rounds thus usually advance at the network's actual latency rather than at worst-case delay bounds.

Mechanism 2: Leader scheduling. Real blockchains reassign leader slots to validators they have observed to perform well: HammerHead [19] assigns more leader slots to the validators whose blocks gathered the most votes under the schedule in force. Barnacle [20] raises the number of leaders per round by one while the fraction of directly decided slots stays high and halves it when the fraction drops. Both read the agreed DAG to change the schedule, yet the schedule determines which slots exist and so what the Decider decides. Unless the two are stratified, agreement is circular. Dugong runs both as one mechanism built on *schedule configurations*. A schedule configuration fixes the slots of each round, their leaders, and an interval, the range of rounds it governs, and its slots are decided under the configuration's schedule, reading slots past the interval only as anchors. The first committed slot past the interval is the *pivot*, and an update rule reads the pivot's causal history and emits the next schedule configuration, which takes effect after the pivot's round (Barnacle) or at the end of the interval, re-deciding the slots in between (HammerHead). Nothing in a schedule configuration's range depends on the next one, so the dependency is well-founded. Safety holds

for any update rule that reads only the pivot's history, with no synchrony or fairness assumption, whereas liveness needs schedules that keep committing (Section 5).

Mechanism 3: Checkpoints. Like Orcaella [12], Dugong adds a second tier of finality, but to any Decider. Validators sign the state they reach by executing what their Decider commits. A quorum of signatures on one state is a *checkpoint certificate*, and a quorum of acknowledgements of a certificate makes the checkpoint *final*. Two certificates of one epoch and height bind one checkpoint, even when the inputs contain forks. The checkpoint tier finalizes every slot the Decider commits, under leader scheduling, and for responsive validators that crash and restart.

Mechanism 4: Crash recovery. A validator writes every block it receives or creates to disk before acting on it. On restart, it rebuilds every round it missed and asks its peers again for what it lacks. It never builds two blocks for the same round, so safety is unchanged. Once the network has stabilized and the correct validators have stopped crashing, the run behaves like a crash-free run whose network stabilizes at that point, so the liveness results of Section 5 apply from then on. Under asynchrony, Agreement holds and Validity holds with probability 1. This covers all validators crashing and restarting, which matters in real-world blockchains: honest validators routinely update their software and reboot while the network is running, with no observable downtime. Occasionally they all need to apply an emergency fix simultaneously to mitigate a live vulnerability. Crash recovery lets the network resume after such a simultaneous restart.

5. Formal Proofs

We formalize Dugong in Lean 4 [24], [26], [27] and prove that every configuration of the Decider implements atomic broadcast against its own fault and network model, and that the mechanisms of Section 4 preserve it. We also formalize the eight published protocols and show that their commit rules are equivalent to configurations of the Decider. The definitions and proofs are about 125,000 lines of Lean, 40,000 of them concrete executions that exercise every definition. The Decider's theorems are stated in files that contain no proofs, and every principal result rests on Lean's three standard axioms. This split reflects how the development was built: AI agents produced most of the proofs, which Lean's kernel checks, while the definitions and theorem statements, the part a reader must trust, are kept separate and reviewed by hand. Appendix A gives the proofs of Theorems 1 and 2 in more detail.

Atomic broadcast. We prove the definition of Section 2 in the two forms of the literature, Sailfish's Definition 1 [21] and Sui et al.'s Definition 2.1 [22]. Agreement and Validity hold for the *reliable* validators: correct validators that eventually receive every block another reliable validator holds. The adversary controls message delays but does not see the common coin. Stake and the deduplication of transactions are not modeled.

Theorem 1 (Atomic broadcast)**.** *Every configuration of the Decider in Table 1 implements atomic broadcast (Section 2) against its own fault and network model, for the reliable validators and under the leader schedules of Lemma 4.*

Only Validity depends on the network model. Prefix consistency, total order, and integrity hold in every execution, whereas agreement additionally requires eventual delivery among the reliable validators. Under partial synchrony, after GST, Validity holds under round-robin in every configuration of the Decider, and under any leader schedule with what we call *bounded anchor chains* (Lemma 4). Under asynchrony, the common coin retroactively schedules leaders, and as long as the adversary cannot predict it, every slot commits with a probability that approaches 1 as the run grows (Lemma 5).

Theorem 2 (Composition)**.** *At every configuration of the Decider the mechanisms of Section 4 compose (all four under partial synchrony and all but responsive liveness under asyncrhony) and implement atomic broadcast against the configuration's fault model.*

6. Implementation

We implement Dugong in Rust [25] on top of the open-source Mysticeti codebase [6]. The implementation comprises roughly 25k lines of Rust across seven crates, including the DAG engine (blocks, round advancement, storage, networking, and synchronization), the Decider, the replica, a discrete-event simulator, and a command-line interface.

Unified Decider. The Decider implements a single commit rule shared across all protocols. Each protocol is specified as a flat configuration structure holding the parameters of Section 3 and two implementation fields: the round-advancement threshold and whether validators wait for leader blocks. In the Rust code, the Decider instantiates one committer per pipelined wave offset and leader slot, evaluates the direct rule followed by the indirect rule on each slot, and outputs the longest decided prefix of leaders. The commit logic contains no protocol-specific branches: each of the twelve configurations evaluated in Section 7 is simply a constructor that checks the committee size bound and sets the corresponding parameters. When $f = 0$, signature generation and verification are disabled, since crashed validators do not forge messages.

Simulator. The simulator runs the unmodified consensus code, including the DAG engine, the Decider, the replica logic, and storage, and replaces only the asynchronous runtime: a thin abstraction swaps Tokio for a deterministic discrete-event scheduler that controls virtual time, task spawning, and message delivery, so that a single random seed determines the execution. Because all simulated validators run on one machine, CPU contention is not representative of a distributed deployment, so we stub out cryptographic operations. Point-to-point links are FIFO with a bounded window of in-flight messages, and per-message latency is drawn either uniformly or from measured inter-region round-trip times, with optional exponential jitter.

Faults are injected at the network layer (Section 7.1). Asynchronous configurations elect leaders via a seeded pseudo-random function instead of a common coin.

7. Evaluation

We evaluate all twelve configurations of the Decider using the same codebase and simulated network environment. We test four claims:

- **C1**: Fault-free commit latency is determined by the number of rounds a configuration requires to commit, rather than by its fault model or quorum size.
- **C2**: Every configuration remains live up to its theoretical fault bound. Under crash faults, the random leader election of asynchronous configurations avoids bursts of consecutive crashed leaders, reducing tail latency at the expense of a higher median latency.
- **C3**: An equivocating leader delays its slot by roughly one wave and blocks subsequent slots, which penalizes long, pipelined waves the most.
- **C4**: Under high network jitter, partially synchronous protocols with a separate certificate round degrade sharply, but two-round protocols and asynchronous protocols with a single buffer round maintain high direct commit rates.

Our experiments evaluate protocol performance under crash faults and leader equivocation. Benchmarking BFT protocols against arbitrary Byzantine strategies is an open research question [28], and we establish worst-case safety and liveness through our mechanized proofs (Section 5).

7.1. Experimental Setup

We denote hybrid configurations by their fault bounds (f, c) and test three operating points for each family: *balanced*, *Byzantine-heavy*, and *crash-heavy*. Specifically, Hybrid Mysticeti and Hybrid Mahi-Mahi are configured at $(9, 9)$, $(13, 3)$, and $(2, 19)$, while Orcaella and Orcaella Async are configured at $(6, 5)$, $(7, 3)$, and $(2, 11)$, matching the ratios of the Orcaella paper [12]. The last two exceed Orcaella's minimum committee and so run a safe variant with a lower skip threshold (Section 3.4).

We simulate a committee of $n = 46$ validators, a network size comparable to prior DAG consensus evaluations [6], [7], [9], [13], [21]. Since $46 = 3 \cdot 15 + 1 = 5 \cdot 9 + 1$, both the $3f + 1$ and $5f + 1$ resilience bounds are tight, and every hybrid bound is met within at most two validators. Link latencies are drawn uniformly between 40 and 80 ms. We denote the average duration of a DAG round by ρ, which is approximately 65 ms without faults or jitter. Because absolute latencies scale with the underlying link delay, we also report latency differences in multiples of ρ. Partially synchronous configurations use a leader timeout of 160 ms, and each validator generates transactions at a constant rate across all runs. When network jitter is enabled, each message experiences an additional delay drawn from an exponential distribution with a mean of 120 ms, capped at 640 ms. Each simulation runs for 120 s, and metrics

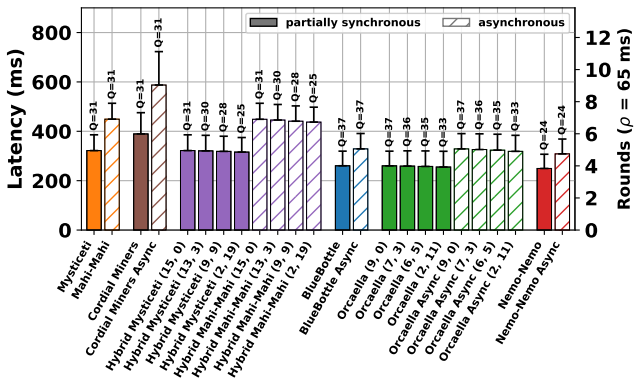


Figure 3. Fault-free latency of every protocol family (p50, whisker to p90; solid: partially synchronous, hatched: asynchronous). Hybrid configurations are shown at all four (f, c) points, from $c = 0$ to crash-heavy, and Q is the round quorum. Right axis: rounds of $\rho = 65$ ms.

are collected after a 30 s warm-up period across all live validators. We measure end-to-end latency from the time a transaction is submitted until it is committed.

To stress-test the protocols, we assign faults to consecutive validator indices: crashed validators occupy the last k indices and equivocating validators occupy the first e indices, so a round-robin schedule encounters all faulty leaders consecutively. Crashed validators are disconnected at the start of the run. Following production implementations, validators skip the leader timeout if their connection to the leader is down, so crashed leaders do not trigger timeout delays (safe, but outside the liveness proofs). In each round where it holds a leader slot, an equivocating validator sends every peer two conflicting blocks with the same content and different digests, in opposite order to odd- and even-indexed peers, so half of the committee sees each block first [28].

For each data point, we report the median across 7 random seeds of the p50 latency, with error bars extending to the p90 latency. Across all ∼3,000 runs of our evaluation campaign (roughly 2,500 vCPU-hours), the simulator's end-of-run safety check never found two honest validators that committed conflicting sequences.

7.2. Fault-Free Latency

Protocols cluster by the number of DAG rounds required to commit rather than by their fault model (Figure 3). In particular, all two-round partially synchronous protocols achieve nearly identical p50 latency (around 255 ms), whether they tolerate crash faults (Nemo-Nemo), Byzantine faults (BlueBottle), or hybrid faults (Orcaella). Similarly, the three-round partially synchronous protocols, Mysticeti and Hybrid Mysticeti, commit approximately 1ρ later.

The round quorum size Q barely matters. Within the hybrid families, moving from $(15, 0)$ to $(2, 19)$ reduces Hybrid Mysticeti's quorum from 31 to 25 while changing its p50 latency by only 0.1ρ, and moving from $(9, 0)$ to $(2, 11)$ reduces Orcaella's quorum from 37 to 33 with the same 0.1ρ difference. The asynchronous hybrid families vary by

at most 0.2ρ. The same holds across protocols: Nemo-Nemo ($Q = 24$) commits within 0.2ρ of BlueBottle ($Q = 37$). At $c = 0$, each hybrid protocol matches the latency of its corresponding pure-Byzantine baseline (Orcaella matches BlueBottle, Hybrid Mysticeti matches Mysticeti, and Hybrid Mahi-Mahi matches Mahi-Mahi) within 1 ms.

Each asynchronous protocol adds latency proportional to its buffer rounds: the three-round asynchronous protocols (BlueBottle Async, Orcaella Async, and Nemo-Nemo Async) commit 1ρ later than their partially synchronous counterparts, and the five-round protocols (Mahi-Mahi and Hybrid Mahi-Mahi) commit 2ρ later, while the common coin itself introduces no measurable overhead. The only exceptions are the unpipelined Cordial Miners variants: because Cordial Miners elects a leader only once per w-round wave, transactions wait an average of $(w-1)/2$ additional rounds before entering a leader wave, placing Cordial Miners 1ρ above Mysticeti and Cordial Miners Async 2ρ above Mahi-Mahi. These observations validate Claim **C1**.

> **Takeaway 1**: *Fault-free latency is governed by the number of rounds in a wave: three-round rules commit 1ρ after two-round rules, and each asynchronous buffer round adds 1ρ, whereas changing the quorum size shifts latency by at most 0.2ρ.*

7.3. Crash Faults

Figure 4 shows latency as the number k of crashed validators grows. Every configuration remains live up to its fault bound and stalls one crash beyond it: 9 crashes for BlueBottle, 13 for Orcaella $(2, 11)$, 15 for Mysticeti, Mahi-Mahi, and Cordial Miners, 21 for Hybrid Mysticeti and Hybrid Mahi-Mahi $(2, 19)$, and 22 for Nemo-Nemo.

Partially synchronous configurations. Crash faults affect performance in two ways: they reduce the number of active voters and they introduce rounds with crashed leaders. With fewer live validators available to form a quorum of size Q, each validator must wait for blocks over slower network links before advancing to the next round. This longer round accounts for the rise in median latency: Mysticeti's p50 stays at 4.7ρ while ρ grows from 68.7 ms to 79.0 ms at its fault bound. In contrast, crashed leaders drive tail latency. Because crashed validators occupy consecutive indices, the round-robin schedule encounters a burst of k consecutive crashed leaders once every 46 rounds. During this burst, no new leader is committed until the first live leader proposes, which delays transactions submitted during that window (increasing p90 latency) while leaving the rest of the schedule unaffected (preserving p50 latency). Each crashed leader in the burst adds roughly one round to the tail: at the fault bound, the p90 latency of every partially synchronous protocol grows by about 1ρ per crashed validator (reaching 13ρ for Mysticeti at $k = 15$).

Asynchronous configurations. Although crashed validators still occupy consecutive indices, the common coin selects each slot's leader uniformly at random. As a result, crashed leaders are distributed across the execution rather than concentrated in a single burst, allowing the protocol to recover within a single wave. For Mahi-Mahi, p90 latency increases by only 2ρ at $k = 15$. By $k = 8$, the asynchronous protocols already exhibit lower tail latency than their partially synchronous counterparts, and at the fault bound their p90 latency is roughly cut in half (for example, 727 ms for Hybrid Mahi-Mahi compared to 1,687 ms for Hybrid Mysticeti at $k = 21$).

The median latency of the asynchronous protocols, however, stays above that of their partially synchronous counterparts by the number of buffer rounds at every k, since the buffer rounds slow down like any other (2ρ between Mahi-Mahi and Mysticeti at $k = 15$, and likewise for the hybrid pair at $k = 21$).

Nemo-Nemo Async and Cordial Miners Async are the only asynchronous configurations whose median latency increases substantially with k, growing by about 6ρ at $k = 22$ and 3ρ at $k = 15$, respectively, compared to 1.3ρ for Mahi-Mahi. Both protocols omit the direct skip rule ($S = n$). Consequently, a slot led by a crashed validator cannot be skipped within its own wave and must wait to be resolved indirectly by a later committed anchor, holding back all subsequent slots in the meantime. Random leader election therefore provides no tail benefit for these two protocols, and the p90 latency of Nemo-Nemo Async matches that of partially synchronous Nemo-Nemo. These results confirm Claim **C2**.

> **Takeaway 2**: *All configurations remain live up to their exact fault bounds. Under partial synchrony, crashes increase median latency only by slowing down rounds, while consecutive crashed leaders increase tail latency by about 1ρ per crash. Random leader election in asynchronous protocols breaks up bursts of crashed leaders and halves tail latency at the fault bound, provided crashed slots can be skipped directly.*

7.4. Byzantine and Hybrid Faults

Figure 5 evaluates performance as e validators equivocate in every round without crashing. When an equivocating validator is elected leader, it splits the committee's votes between its two conflicting blocks, preventing the slot from reaching either the direct commit quorum or the direct skip quorum [28]. The fraction of directly committed leaders therefore drops to $1 - e/n$ (0.68 for Mysticeti at $e = 15$), and each equivocating slot can only be resolved indirectly by an anchor at least one wave later, delaying the slot by roughly w rounds and blocking the $w - 1$ slots behind it.

Equivocators only. For partially synchronous protocols, the latency curves resemble those under crash faults: at $e = 15$, Mysticeti's p50 latency increases by 0.6ρ while its p90 latency increases by about 13ρ. Among the three-round asynchronous protocols, random leader election again yields a slightly higher p50 latency in exchange for a lower p90 latency (for BlueBottle at $e = 9$, the asynchronous p90

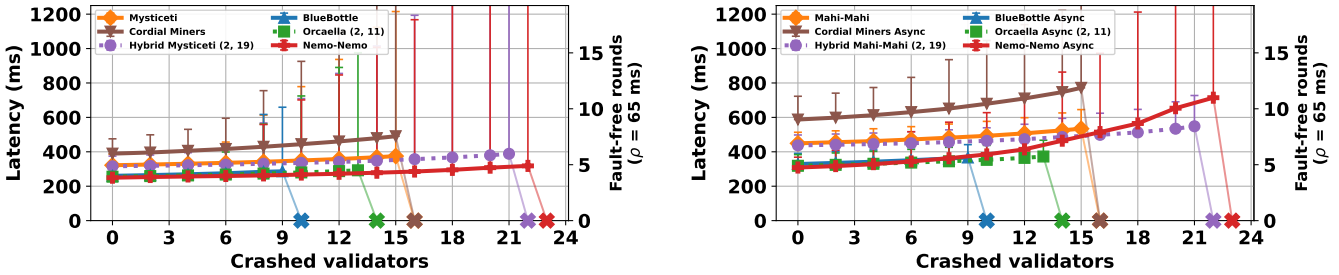


Figure 4. Latency as validators crash, for partially synchronous (left) and asynchronous (right) configurations (p50, whisker to p90). An X marks the first crash count at which a configuration stalls. Hybrid configurations are shown at their crash-heavy point. Right axis: fault-free rounds of $\rho = 65$ ms. Rounds lengthen to about 79 ms at each fault bound. Whiskers above 1,250 ms are clipped.

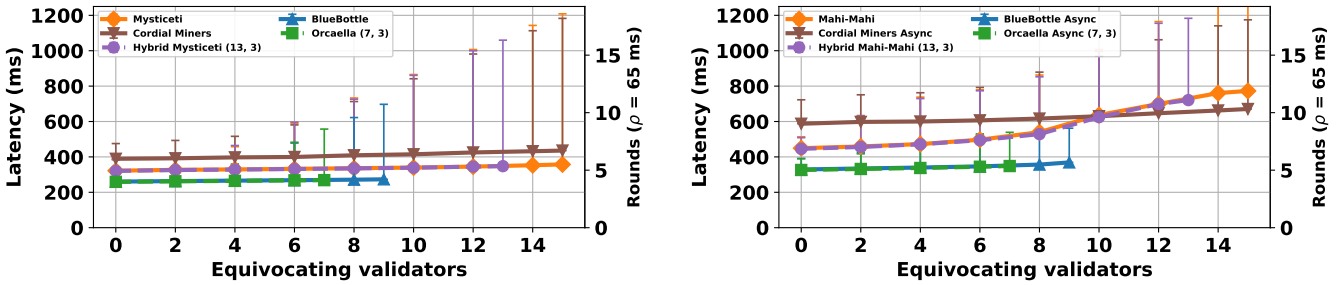


Figure 5. Latency as validators equivocate, with no crashes, for partially synchronous (left) and asynchronous (right) configurations (p50, whisker to p90). Hybrid configurations are shown at their Byzantine-heavy point. Right axis: rounds of $\rho = 65$ ms. Whiskers above 1,250 ms are clipped.

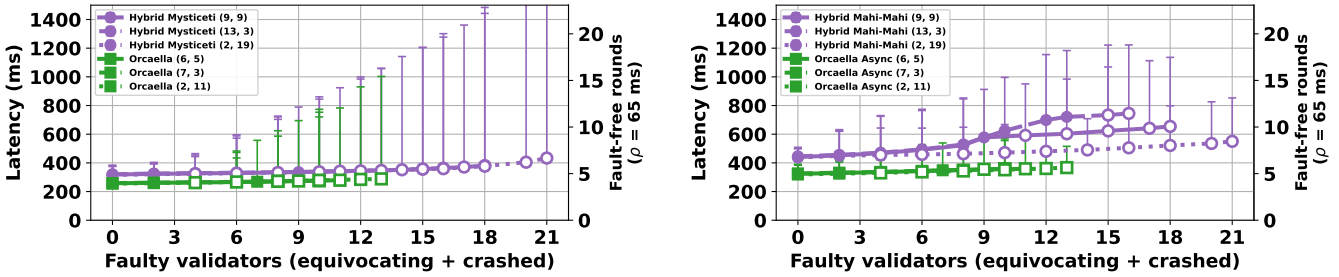


Figure 6. Latency of the hybrid configurations against the total number of faulty validators, for partially synchronous (left) and asynchronous (right) configurations (p50, whisker to p90). Each line first adds equivocators up to f (filled markers), then crashes up to c (hollow markers). Line style marks the (f, c) point: solid balanced, dashed Byzantine-heavy, dotted crash-heavy. Right axis: fault-free rounds of $\rho = 65$ ms. Whiskers above 1,500 ms are clipped.

is 2ρ lower than the partially synchronous one). In contrast, the five-round asynchronous protocols experience a sharp increase in median latency as well: Mahi-Mahi's p50 grows by 5ρ at $e = 15$, and Hybrid Mahi-Mahi $(13, 3)$ exhibits the same trend.

Near $e = 10$, Mahi-Mahi's latency exceeds that of Cordial Miners Async because pipelining becomes a disadvantage under frequent equivocation. Mahi-Mahi schedules a leader in every round, so when $e/n = 1/3$ of leaders equivocate and stall for a five-round wave, nearly every transaction is queued behind an undecided slot. Cordial Miners Async, by contrast, elects only one leader per five-round wave, so an equivocating leader delays only its own wave. The two curves cross at the point where this head-of-line blocking outweighs Mahi-Mahi's 2ρ fault-free advantage.

Hybrid faults. Figure 6 first introduces f equivocating validators and then adds up to c crashed validators. Under partial synchrony, the three (f, c) configurations of Hybrid Mysticeti and Orcaella follow nearly the same trajectory, because replacing crashes with equivocators produces two opposing effects: crashed validators stop voting and lengthen every round, whereas equivocating validators continue voting and keep rounds fast, but an equivocating leader stalls its slot until the next anchor instead of being directly skipped. At the same total fault count, an equivocator adds 1–2ρ more to the p90 latency than a crash. Under asynchrony, where the common coin scatters faulty leaders, the three Hybrid Mahi-Mahi configurations separate clearly, with the crash-heavy configuration achieving the lowest latency (at 16 total faults, its p50 is 3.7ρ below the Byzantine-heavy configuration) because crashed leaders are directly skipped within their wave while equivocating leaders wait for a later

anchor. This confirms Claim **C3**.

> **Takeaway 3**: *An equivocating leader prevents both direct commit and direct skip, stalling its slot for a full wave and blocking subsequent slots. Protocols with waves of up to three rounds keep p50 latency within 1ρ at f equivocators, whereas five-round pipelined protocols suffer severe head-of-line blocking and eventually fall behind unpipelined waves.*

7.5. Network Jitter

Figure 8 introduces exponential message jitter without validator faults, increasing the average round duration to $\rho \approx 300\,\text{ms}$. The protocols fall into three groups. First, all asynchronous configurations continue to commit nearly every leader directly (at least 98% across all runs, and 100% in 65 of the 70 runs), so their latency increases only because each round is longer: Mahi-Mahi commits at a p50 of 6.9ρ, matching its round count without jitter. Second, the two-round partially synchronous protocols (BlueBottle, Orcaella, and Nemo-Nemo) directly commit 77–81% of their leaders and require roughly one additional round at the median (4.7–5.3ρ compared to 3.9ρ without jitter). Third, the three-round partially synchronous protocols, which require a separate certificate round (Mysticeti, Cordial Miners, and Hybrid Mysticeti), degrade severely: their direct commit rate drops to 7–42%, and their p50 latency rises to 10–50 rounds. The three-round asynchronous protocols do not suffer this degradation because they use a two-round commit rule preceded by one buffer round rather than a certificate round. The most affected configurations (Mysticeti, Cordial Miners, and Hybrid Mysticeti $(13, 3)$) do not stabilize within the 120 s window, with goodput ratios of 0.4 to 1.2 across seeds, so their bars reflect order-of-magnitude behavior.

Impact on the certificate round. Under a three-round commit rule, a direct commit must assemble a quorum twice: each block in round $r + 2$ must reference Q votes from round $r + 1$ to form a certificate, and the validator must collect Q such certificates. In contrast, a two-round rule requires only Q votes in round $r + 1$. Under jitter, enough blocks in round $r + 1$ still vote for the leader (70–83% across all configurations), which suffices for a two-round rule. However, because validators advance to round $r+2$ as soon as they receive Q blocks from round $r+1$, a typical round-$(r+2)$ block references only about 36 blocks from round $r + 1$, of which roughly 29 are votes, two short of Mysticeti's threshold $Q = 31$. Consequently, each slot accumulates an average of only 18 certificates instead of the required 31, yielding a direct commit rate of 0.11 (matching the 0.12 rate predicted from the per-round vote distribution). The Hybrid Mysticeti configurations collect a similar number of certificates per slot but require fewer certificates per slot, $2f + c + 1 = 24$ to 30 instead of 31, which accounts for the $3\times$ spread in their latencies.

Two-round $5f+1$ protocols also differ from Mysticeti in their parent quorum size (37 parents per block for BlueBottle versus 31 for Mysticeti). In Appendix B, we isolate these two factors through an ablation study and show that the larger parent quorum accounts for most of BlueBottle's resilience to jitter, while eliminating the second voting round saves about one round of latency.

Buffer rounds. Figure 7 compares each asynchronous family across two wave lengths w under network jitter. Adding a buffer round improves latency only when it recovers lost direct commits. Increasing the wave length from $w = 2$ to $w = 3$ has this effect: at $w = 2$, Nemo-Nemo Async directly commits only 78% of its leaders (73% with crashes), forcing the remaining slots to wait for a subsequent anchor. Restoring those direct commits at $w = 3$ offsets the extra buffer round at the median, leaving p50 latency unchanged both with and without crashes, and significantly reduces tail latency (lowering p90 by 0.5 s without crashes and by 1.4 s with crashes). For $w \geq 3$, however, all asynchronous configurations already directly commit at least 97% of their leaders, so adding another buffer round (from $w = 3$ to $w = 4$ for Orcaella Async, or from $w = 4$ to $w = 5$ for Mahi-Mahi and Hybrid Mahi-Mahi) provides no further gain in direct commits and simply adds 1ρ to the median latency. This confirms Claim **C4**.

> **Takeaway 4**: *Under network jitter, first-round votes still form a quorum, but second-round certificates often fall short: three-round partially synchronous protocols directly commit only 7–42% of leaders and require 10–50 rounds, whereas two-round protocols directly commit 77–81% at the cost of roughly one extra round. Asynchronous protocols directly commit over 97% of leaders with a single buffer round, which adds no median latency, while any further buffer round adds 1ρ.*

8. Related Work

DAG-based consensus and leader scheduling. Certified DAG protocols, such as DAG-Rider [8], Narwhal and Tusk [7], Bullshark [9], Sailfish [21], and Shoal++ [29], disseminate each block by reliable or consistent broadcast. The resulting certificate rules out equivocation and ensures availability, at the cost of extra message delays and signatures in every round. Certified DAGs are deployed in Aptos and Aleo. Uncertified DAG protocols, which build on ideas from Hashgraph [3] and Blockmania [4] and were developed in Cordial Miners [5] and Mysticeti [6], instead tolerate equivocation within the DAG, and each defines a standalone commit rule and proof, whereas Dugong captures their shared decision logic in a single parameterized Decider. For leader scheduling, Carousel [18] introduces reputation-based leader selection for chained BFT and HammerHead [19] brings it to certified DAGs. Dugong composes scheduling with every configuration of the Decider (Section 4).

Systematizations and unified BFT frameworks. For leader-based BFT, Bedrock [30] defines a parameterized design space and execution platform for partially synchronous

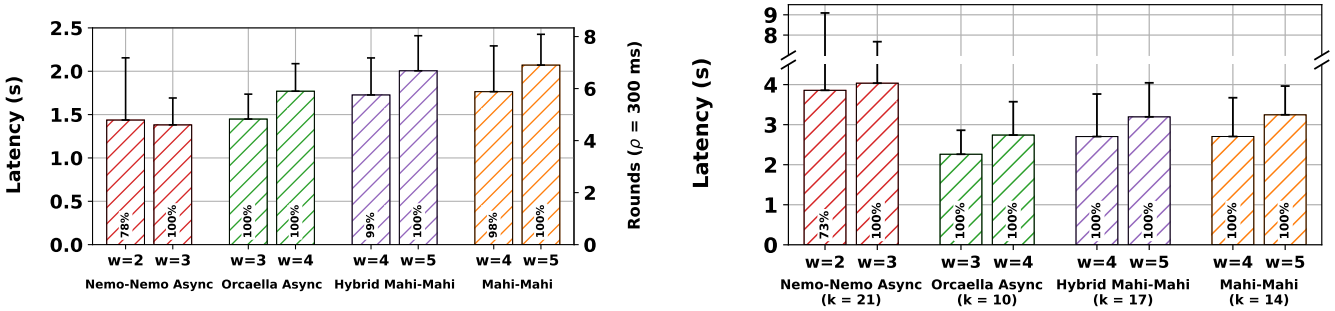


Figure 7. Latency of each asynchronous family at two wave lengths w under jitter, without (left) and with (right) crashes, k one below each fault bound (p50, whisker to p90). The right panel is broken at 4.5 s. The number at the foot of each bar is its direct commit rate. Right axis (left panel): rounds of $\rho \approx 300$ ms.

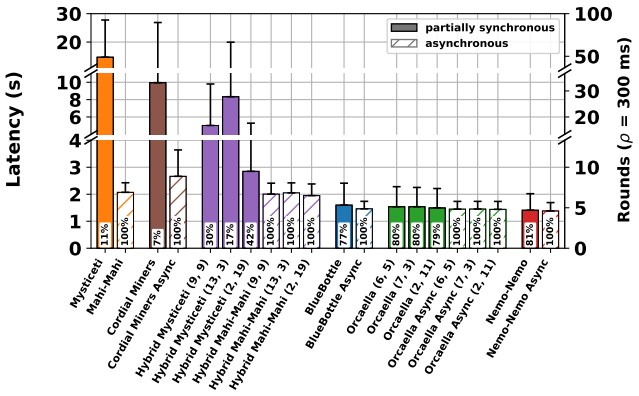


Figure 8. Latency under message jitter without faults (p50, whisker to p90). The axis is broken at 4 and 11 s. The number at the foot of each bar is its direct commit rate. Right axis: rounds of $\rho \approx 300$ ms.

protocols. For DAG consensus, Raikwar et al. [31] classify protocols by consistency and availability trade-offs, but provide no common decision rule, proof, or implementation. Dugong provides a parameterized protocol framework and unified evaluation platform for uncertified DAG consensus, and pairs every configuration and feature mechanism with machine-checked proofs.

Mechanized consensus verification. Mechanized proofs of leader-based BFT establish both safety and liveness of Jolteon in Coq through LiDO [32]. Bythos [33] provides compositional Coq proofs of safety and liveness for BFT protocols assembled from broadcast building blocks, whereas Dugong composes feature mechanisms around a parameterized DAG Decider (Theorem 2). For DAG consensus, Bertrand et al. [34] verify safety, though not liveness, for DAG-Rider, Cordial Miners, Hashgraph, Bullshark, and an Aleph variant in TLA+ by separating DAG construction from ordering to reuse invariants. LiDO-DAG [35] proves safety and liveness for Narwhal, Bullshark, and Sailfish in Rocq, and Qiu et al. [36] extend LiDO-DAG to Mysticeti and identify a liveness bug caused by validators that jump rounds without building blocks. All of these DAG formalizations verify individual protocols at $n \geq 3f + 1$ under Byzantine faults alone. Within Lean 4 [26], Veil [37] combines SMT automation with interactive proofs to verify transition systems such as Paxos variants, reliable broadcast, and Rabia, none of which is a DAG protocol. In contrast, Dugong formalizes uncertified DAG consensus directly in Lean 4 and Mathlib [27] and proves both safety and liveness for one parameterized Decider across partial synchrony and asynchrony as well as crash, Byzantine, and hybrid fault models, together with its feature compositions.

9. Conclusion

Many uncertified DAG protocols are in fact one protocol. We show that a single parameterized Decider recovers eight published protocols across network and fault models and yields four new ones. As a result, one proof covers every configuration, mechanism, and composition, and one implementation runs them all. Choosing a protocol reduces to trading fault tolerance against latency, at equal throughput, by changing parameters only. A blockchain therefore need not commit early to one protocol or implementation.

Results not presented here. Formalizing an entire family of protocols, rather than a single one, is a substantial effort: our Lean codebase spans about 125,000 lines. It produced more results than fit here, all available in our formalization [24]. Besides the eight protocols that the Decider recovers, we have formalized Bluestreak [38] and FinWhale [39]. While developing Dugong, we found attacks against Minnow [16] and Black Marlin [15]. Beyond the proofs about the Decider's specification, we also prove that a concrete Lean implementation of the Decider conforms to that specification. Finally, this paper omits rate limiting, garbage collection, regenesis, and block synchronization: the formalization handles them, and denial of service, from the ground up through *k-partial DAGs*, in which a block is valid without its full causal history.

Open problem. Our Rust implementation is not formally linked to the Lean specification or implementation. With mature theorem provers and AI agents that can produce proofs, machine-checked formalization of whole protocol families is becoming cheap, but connecting such proofs to production code remains open.

References

[1] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *J. ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[2] C. Dwork, N. A. Lynch, and L. J. Stockmeyer, "Consensus in the presence of partial synchrony," *J. ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[3] L. Baird and A. Luykx, "The Hashgraph protocol: Efficient asynchronous BFT for high-throughput distributed ledgers," in *Proc. COINS*. IEEE, 2020, pp. 1–7.

[4] G. Danezis and D. Hrycyszyn, "Blockmania: from block DAGs to consensus," *CoRR*, vol. abs/1809.01620, 2018.

[5] I. Keidar, O. Naor, O. Poupko, and E. Shapiro, "Cordial Miners: Fast and efficient consensus for every eventuality," in *Proc. DISC*, ser. LIPIcs, R. Oshman, Ed., vol. 281. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023, pp. 26:1–26:22.

[6] K. Babel, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, A. Koshy, A. Sonnino, and M. Tian, "Mysticeti: Reaching the latency limits with uncertified DAGs," in *Proc. NDSS*. The Internet Society, 2025.

[7] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus," in *Proc. EuroSys*, Y. Bromberg, A. Kermarrec, and C. Kozyrakis, Eds. ACM, 2022, pp. 34–50.

[8] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is DAG," in *Proc. ACM PODC*. ACM, 2021, pp. 165–175.

[9] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: DAG BFT protocols made practical," in *Proc. ACM CCS*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM, 2022, pp. 2705–2718.

[10] P. V. Vos, A. Sonnino, G. Tsimos, P. Jovanovic, and L. Kokoris-Kogias, "Bluebottle: Fast and robust blockchains through subsystem specialization," *CoRR*, vol. abs/2511.15361, 2025.

[11] R. Kerur, P. Tennage, P. Jovanovic, D. Malkhi, A. Sonnino, and I. Zablotchi, "Finding nemo-nemo: CFT dag-based consensus in the WAN," *CoRR*, vol. abs/2604.08914, 2026.

[12] L. Kokoris-Kogias and A. Sonnino, "Orcaella: Hybrid fault tolerance with client-selectable finality latency," *CoRR*, vol. abs/2607.04789, 2026.

[13] P. Jovanovic, L. Kokoris-Kogias, B. Kumara, A. Sonnino, P. Tennage, and I. Zablotchi, "Mahi-Mahi: Low-latency asynchronous BFT DAG-based consensus," in *Proc. IEEE ICDCS*. IEEE, 2025, pp. 549–559.

[14] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[15] I. Amores-Sesar, V. Grøndal, A. Holmgård, and M. Ottendal, "DAG it off: Latency prefers no common coins," in *Proc. DISC*, ser. LIPIcs, vol. 356. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, pp. 5:1–5:17.

[16] P. Kuznetsov, M. Perion, and S. Tucci-Piergiovanni, "Minimizing commit rules for DAG-based atomic broadcast," *CoRR*, vol. abs/2608.18029, 2026.

[17] S. Blackshear, A. Chursin, G. Danezis, A. Kichidis, L. Kokoris-Kogias, X. Li, M. Logan, A. Menon, T. Nowacki, A. Sonnino, B. Williams, and L. Zhang, "Sui Lutris: A blockchain combining broadcast and consensus," in *Proc. ACM CCS*. ACM, 2024, pp. 2606–2620.

[18] S. Cohen, R. Gelashvili, E. Kokoris-Kogias, Z. Li, D. Malkhi, A. Sonnino, and A. Spiegelman, "Be aware of your leaders," in *Proc. FC*, ser. Lecture Notes in Computer Science, vol. 13411. Springer, 2022, pp. 279–295.

[19] G. Tsimos, A. Kichidis, A. Sonnino, and L. Kokoris-Kogias, "Hammerhead: Leader reputation for dynamic scheduling," in *Proc. IEEE ICDCS*. IEEE, 2024, pp. 1377–1387.

[20] G. Danezis, Z. De Angeli, A. I. Vitanov, P. Jovanovic, L. Kokoris-Kogias, A. Sonnino, P. Tennage, and I. Zablotchi, "Barnacle: Adaptive multi-leader scheduling for DAG-based consensus," 2026.

[21] N. Shrestha, R. Shrothrium, A. Kate, and K. Nayak, "Sailfish: Towards improving the latency of DAG-based BFT," in *Proc. IEEE S&P*, M. Blanton, W. Enck, and C. Nita-Rotaru, Eds. IEEE, 2025, pp. 1928–1946.

[22] X. Sui, X. Wang, and S. Duan, "Signature-free atomic broadcast with optimal $O(n^2)$ messages and $O(1)$ expected time," in *Proc. IEEE S&P*. IEEE, 2025, pp. 1547–1565.

[23] P. V. Vos, "Odontoceti: Ultra-fast DAG consensus with two round commitment," *CoRR*, vol. abs/2510.01216, 2025.

[24] "lean-dag: A Lean 4 formalization of uncertified DAG consensus," https://github.com/gdanezis/lean-dag (commit `48aa36c`), 2026.

[25] "Dugong: A Rust implementation and discrete-event simulator of uncertified DAG consensus," https://github.com/asonnino/mysticeti (commit `ce84120`), 2026.

[26] L. de Moura and S. Ullrich, "The Lean 4 theorem prover and programming language," in *Proc. CADE*, ser. Lecture Notes in Computer Science, A. Platzer and G. Sutcliffe, Eds., vol. 12699. Springer, 2021, pp. 625–635.

[27] The mathlib Community, "The Lean mathematical library," in *Proc. ACM CPP*. ACM, 2020, pp. 367–381.

[28] S. Bano, A. Sonnino, A. Chursin, D. Perelman, Z. Li, A. Ching, and D. Malkhi, "Twins: BFT systems made robust," in *Proc. OPODIS*, ser. LIPIcs, vol. 217. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 7:1–7:29.

[29] B. Arun, Z. Li, F. Suri-Payer, S. Das, and A. Spiegelman, "Shoal++: High throughput DAG BFT can be fast and robust!" in *Proc. USENIX NSDI*, T. A. Benson and R. N. Mysore, Eds. USENIX Association, 2025, pp. 813–826.

[30] M. J. Amiri, C. Wu, D. Agrawal, A. E. Abbadi, B. T. Loo, and M. Sadoghi, "The bedrock of byzantine fault tolerance: A unified platform for BFT protocols analysis, implementation, and experimentation," in *Proc. USENIX NSDI*. USENIX Association, 2024, pp. 371–400.

[31] M. Raikwar, N. Polyanskii, and S. Müller, "SoK: Dag-based consensus protocols," in *Proc. IEEE ICBC*. IEEE, 2024, pp. 1–18.

[32] L. Qiu, Y. Kim, J. Shin, J. Kim, W. Honoré, and Z. Shao, "Lido: Linearizable byzantine distributed objects with refinement-based liveness proofs," *Proc. ACM Program. Lang.*, vol. 8, no. PLDI, pp. 1140–1164, 2024.

[33] Q. Zhao, G. Pîrlea, K. Grzeszkiewicz, S. Gilbert, and I. Sergey, "Compositional verification of composite byzantine protocols," in *Proc. ACM CCS*. ACM, 2024, pp. 34–48.

[34] N. Bertrand, P. Ghorpade, S. Rubin, B. Scholz, and P. Subotić, "Reusable formal verification of DAG-based consensus protocols," in *Proc. NFM*, ser. Lecture Notes in Computer Science, A. Dutle, L. R. Humphrey, and L. Titolo, Eds., vol. 15682. Springer, 2025, pp. 138–158.

[35] L. Qiu, J. Xiao, J. Shin, and Z. Shao, "LiDO-DAG: A framework for verifying safety and liveness of DAG-based consensus protocols," *Proc. ACM Program. Lang.*, vol. 9, no. PLDI, pp. 1392–1416, 2025.

[36] L. Qiu, J. Xiao, and Z. Shao, "Mechanized safety and liveness proofs for the Mysticeti consensus protocol under the LiDO-DAG framework," in *Proc. IEEE S&P*. IEEE, 2026, pp. 1842–1861.

[37] G. Pîrlea, V. Gladshtein, E. Kinsbruner, Q. Zhao, and I. Sergey, "Veil: A framework for automated and interactive verification of transition systems," in *Proc. CAV*, ser. Lecture Notes in Computer Science, R. Piskac and Z. Rakamarić, Eds., vol. 15933. Springer, 2025, pp. 26–41.

[38] N. Polyanskii, I. Vorobyev, and S. Müller, "Bluestreak: Uncertified DAG BFT with sparse metadata," *IACR Cryptol. ePrint Arch.*, vol. 2026, p. 898, 2026.

[39] R. Ladelsky and R. Friedman, "FinWhale: An optimally resilient 2 rounds terminating DAG protocol," in *Proc. DISC*, ser. LIPIcs, vol. 397. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2026, pp. 2:1–2:22.

Appendix A.
Security Analysis of the Decider

We prove that every configuration of the Decider implements atomic broadcast against its own fault model, as defined in Section 2. Every result holds at every row of Table 1, at every committee from the row's bound, and at every vote rule: any deterministic choice of one block of the slot from the voter's history at and above the slot's round, whenever one exists (Section 3.1). All proofs are mechanized in Lean 4, and we give the argument of each.

Lemma 1 (Total order)**.** *If two correct validators deliver blocks b and b', they deliver them in the same order.*

Proof sketch. We first show that no two views decide a slot differently. Each inequality of Section 3.4 closes one case of comparing two decisions of the slot, by one of the counts of that section. The vote rule enters only through two facts: a block votes for at most one block of a slot, and it never blames a slot it votes in. The two Nemo-Nemo rows violate the inequality against conflicting blocks of one leader, which is needed only when a validator equivocates, and they meet its crash-only weakening. TRYDECIDE outputs exactly the decisions its view derives, at consecutive slots, so two correct validators' sequences of committed leaders are prefixes of one another. Restarting the loop at the last consumed round outputs what one call on the final view would (Section 3.1). Each committed leader delivers the blocks of its causal history in a fixed order, by round and then by identifier, keeping the first block of each round and author not yet delivered, so the delivered sequences are prefixes of one another too. □

Lemma 2 (Integrity)**.** *Every correct validator delivers at most one block of each round and author, and no block twice. Since blocks are signed, a delivered block of a correct validator was built by that validator.*

Proof sketch. Two blocks of a Byzantine author for one round can both lie in a committed leader's history, since blocks are not reliably broadcast. Delivery therefore keeps only the first block of each round and author, so no round and author is delivered twice, and hence no block. Every delivered block is a block of the DAG. □

Lemma 3 (Agreement)**.** *If a reliable validator delivers a block b, every reliable validator eventually delivers b.*

Proof sketch. Under either network model, a reliable view is eventually contained in every other reliable view. The delivered sequence on a larger view extends the one on a smaller view (Lemma 1), so every reliable validator eventually delivers b. □

Lemma 4 (Validity under partial synchrony)**.** *After GST, every slot a correct validator leads commits on every correct view. A block of a reliable validator that every reliable validator holds at round t is delivered by every reliable validator under any schedule with* bounded anchor chains*, in which every slot is decided, directly or through the indirect rule, within a bounded number of rounds. Round-robin gives bounded anchor chains when $W\,(n-|T|) < n$, T the reliable set, and at Hybrid Mysticeti's committee when crashed validators are silent. When every correct validator is reliable, $n-|T| = f+c$, and one of the two conditions holds in every partially synchronous configuration of the Decider: the first everywhere except Hybrid Mysticeti with $c > 0$, and the second there.*

Proof sketch. After GST a correct leader's proposal reaches every correct validator before its timeout expires, so the correct validators build and vote on the wave, and the slot then commits on every view that holds the wave. The scan from round $t+1+W$ reaches a committed leader, and every slot below it is decided: the slots at rounds from $t+2$ by their own scans, and those below through the indirect rule. Through weak references, every reliable block built after round t reaches every block the reliable validators held at t, so the block is in that leader's history. □

Lemma 5 (Validity under asynchrony)**.** *At every asynchronous row, against an adversary that cannot predict the common coin, a slot followed by M waves stays undecided on some correct view with probability at most $(1-s/n)^M$, $s \geq 1$ the number of leaders a wave makes safe (Section 3.4), and every block of a correct validator is delivered by every correct validator with probability 1.*

Proof sketch. A leader is *safe to commit*, or *safe*, if every vote-round block of its wave reaches its proposal. Every correct validator votes for a safe leader's proposal with no timing assumption, so a coin that names a safe leader commits its slot. Since every asynchronous row has $2q \geq n+f+1$, the correct validators share a common core after two rounds, which makes $s \geq 1$ leaders safe. A wave is a fresh, independent draw of the coin, which gives the bound. A block of a correct validator eventually reaches every correct validator, whose next block references it, weakly if nothing else reaches it (Section 2). It is therefore in the history of every later committed safe leader, and is delivered once a safe leader commits, which happens with probability 1. □

Proof of Theorem 1. Theorem 1 follows from Lemmas 1 to 5. The lemmas prove both forms of atomic broadcast in the literature, Sailfish's Definition 1 [21] and Sui et al.'s Definition 2.1 [22]. The forms differ mainly in two clauses: Sailfish's Integrity counts deliveries per round and author, while Sui et al.'s counts messages and requires that some replica broadcast them; and Sui et al.'s Validity requires only that the broadcaster itself deliver its message. Integrity and Total order hold in every execution, while Agreement

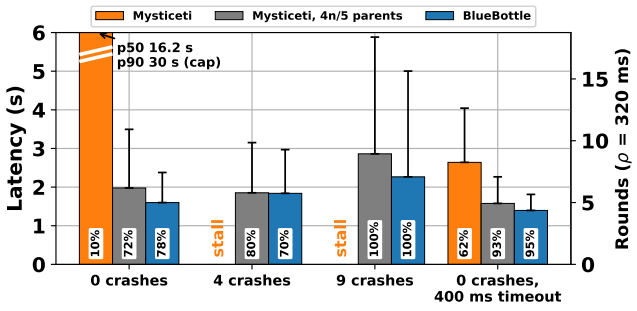


Figure 9. Parent-quorum ablation under jitter (p50, whisker to p90). Plain Mysticeti without crashes runs off the frame, and its values are printed next to it. Mysticeti with $4n/5$ parents is an ablation, not a protocol: it is not live beyond $n/5$ crashes. The number at the foot of each bar is its direct commit rate. Its p90 is capped at 30 s. Right axis: rounds of $\rho \approx 320$ ms, and at 9 crashes, rounds last about 620 ms. Separate campaign from the other figures.

and Validity hold for the reliable validators. Each published row equals our formalization of its protocol, up to the vote rule (Section 3.3). Theorem 1 therefore holds for these formalizations of the published protocols.

Proof of Theorem 2. A scheduled run is a sequence of leader configurations, each in force over a range of rounds that ends at the next configuration's start. Its *ledger* is the sequence of leaders the run commits in these ranges, and a validator delivers the causal histories of the ledger's leaders, through references and weak references, as in Lemma 1. Integrity holds of the delivered sequence of any ledger, by Lemma 2. Validators decide every slot alike, so the next configuration, a function of committed history, is the same on every view, and two validators' ledgers up to configurations both have closed are prefixes of one another. Total order and Agreement follow as in Lemmas 1 and 3. For Validity, let configuration k be in force from round s_k, with interval I, and let it be good from $s_k + \delta$, $\delta \geq 1$: its schedule has bounded anchor chains with commit gap g, and the reliable validators run it responsively, with crashes and restarts, past GST and the last restart. If $\delta + g \leq I$, a leader of round at least $s_k + \delta$ commits before the range ends, so it is in the ledger. Configurations start at least one round apart, so $s_{t+2} \geq t+2$, and that leader's round exceeds $t+1$. Through weak references, its history holds every block the reliable validators held at round t. Under asynchrony every configuration is good when the coin names a wave of safe leaders often enough, which fails with probability decaying geometrically in the number of waves, as in Lemma 5. Under HammerHead, the coin draws from the reputation distribution, whose support must meet every set of validators the adversary can make safe. Checkpoints finalize every slot a closed configuration commits, since they read the committed leaders.

Appendix B. Why Two-Round Protocols Tolerate Jitter

Section 7.5 leaves open why two-round $5f + 1$ configurations tolerate message jitter that slows Mysticeti down to tens of rounds. Two differences between the protocols may contribute: (1) $5f + 1$ configurations wait for a larger parent quorum in every round (BlueBottle advances on $4n/5 + 1 = 37$ blocks, compared to 31 for Mysticeti), and (2) they commit after a single voting round instead of requiring a second round of certificates. To separate these two effects, we evaluate an ablation that runs Mysticeti with a parent quorum of 37 blocks while keeping all of its other parameters unchanged. This configuration preserves Mysticeti's safety but loses liveness beyond $n/5 = 9$ crashes, serving purely as a diagnostic baseline. Because the ablation and BlueBottle share the same parent quorum and leader timeout, they construct the same underlying DAG: their vote counts match seed for seed with crashes, and their vote shares differ by at most 0.03 without crashes. Any latency difference between them therefore isolates the effect of the commit rule. We also re-run unmodified Mysticeti and BlueBottle in the same campaign as baselines.

Effect of a larger parent quorum. Figure 9 shows that increasing the parent quorum size accounts for most of the latency difference. Without crashes, Mysticeti's direct commit rate increases from 0.10 to 0.72 and its p50 latency drops from roughly 50ρ to within 1ρ of BlueBottle. With 4 and 9 crashes, unmodified Mysticeti stalls and commits no leaders during the 120 s window, whereas the ablation continues to commit smoothly. (This stall does not violate Mysticeti's theoretical liveness guarantee, which holds only once the leader timeout exceeds message delays, whereas jittered delays reach 720 ms compared to the 160 ms timeout.) The improvement comes from the higher density of certificates rather than of votes. Specifically, the fraction of round-$(r+1)$ blocks that vote for the leader barely changes (0.84, compared to 0.81 for unmodified Mysticeti), but each round-$(r+2)$ block now references roughly 39 blocks from round $r+1$ instead of 36. As a result, a typical round-$(r+2)$ block carries about 32 votes instead of 29, clearing Mysticeti's certificate threshold $Q = 31$. The share of round-$(r+2)$ blocks that qualify as certificates increases from 39% to 73%, and each slot collects an average of 34 certificates instead of 18.

Effect of a single voting round. The remaining difference between the ablation and BlueBottle reflects the second factor: committing in one voting round rather than two. At 9 crashes, the 37 remaining live validators match the parent quorum size, so every block references every live block from the previous round, and both rules directly commit every live leader and directly skip every crashed leader. In this regime, the latency difference is exactly the one-round difference in wave length, 1ρ (with $\rho \approx 620$ ms). Without crashes, the difference between the two is also approximately 1ρ.

At 4 crashes, the latency difference between the ablation and BlueBottle disappears because the two commit rules

respond to crashes in opposite ways. BlueBottle requires a fixed threshold of 37 votes from a smaller set of 42 live validators, so the fraction of slots that reach 37 votes drops from 79% (with no crashes) to 68%, reducing its direct commit rate to 70%. In contrast, the ablation's 37 parents cover a larger fraction of the 42 live blocks in round $r+1$, so each round-$(r+2)$ block carries more votes: the fraction of slots that reach the required 31 certificates rises from 69% (with no crashes) to 80%. The extra indirect commits incurred by BlueBottle offset the one-round advantage of its shorter wave.

Effect of a longer leader timeout. Increasing the leader timeout to 400 ms also increases the number of votes observed by each decision-round block. While a 400 ms timeout restores progress for unmodified Mysticeti (raising its direct commit rate to 0.62), the larger parent quorum achieves a higher direct commit rate (0.93, close to BlueBottle's 0.95) and commits roughly 3ρ faster.

Appendix C. Safety Violations in Black Marlin and S-Minnow

Black Marlin [15] and S-Minnow [16] are partially synchronous uncertified DAG protocols that commit from less of the DAG than Mysticeti at $n \geq 3f+1$. Both decide a leader slot by a direct rule, which commits a block that a quorum of the round above references, and an indirect rule, which decides earlier slots once a later leader commits and tests only whether a block of the slot lies in that leader's history. When the leader equivocates, the two rules can choose different blocks. Figures 10 and 11 give an execution of each at $n = 4,\ f = 1$.

Black Marlin. elects one anchor per round. Its direct rule (line 16 of Algorithm 1 [15]) commits the anchor of round $r-2$ when it has $n-f$ supporters and a supported anchor of round $r-1$ references it; a validator that fails the test never retries the round. The commit recursion then delivers the anchor's history, recursing on one anchor per round chosen by a metric that ignores support, and skips any block whose author and round it already delivered.

S-Minnow. commits a leader block l when $2f+1$ validators reference it from the round above, and every earlier slot holds a block that lies in l's history, or is concurrent with l and committed or skipped (Definition 9 of [16]). The first case clears a slot without deciding it.

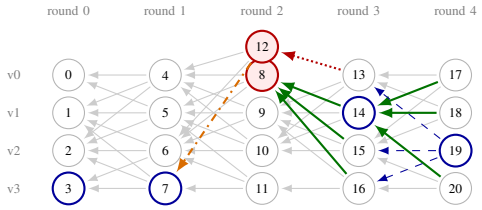


Figure 10. The Black Marlin execution at $n = 4$, $f = 1$, with validator 0 Byzantine. Blue rings mark anchors, red rings the equivocating anchor blocks 8 and 12 of round 2. Blocks 14, 15, 16 reference 8, and 17, 18, 20 reference the round-3 anchor 14 (green), so validator X, holding them, commits 8 by the direct rule. Validator Y lacks 17, 18, 20 and fails the test. The round-4 anchor 19 omits 14 (blue, dashed); rounds 5 and 6, not drawn, commit 19 at Y, whose recursion drops to round 2 and prefers 12, which reaches the round-1 anchor 7 (orange), over 8. X outputs 8, Y outputs 12, and line 27 makes each drop the other. Blocks 5 and 7, of correct authors, come out in opposite orders.

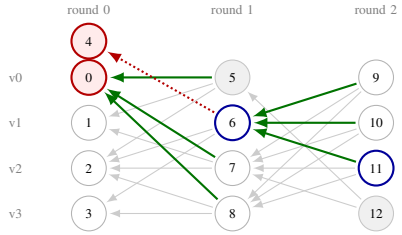


Figure 11. The S-Minnow execution at $n = 4$, $f = 1$, with validator 0 Byzantine and one leader per round by round-robin. Validator 0 equivocates at round 0 with blocks 0 and 4. Blocks 5, 7, 8 reference 0 (green), so 0 carries a quorum in the whole DAG; only the round-1 leader 6 references 4 (red). A correct validator's view lacks 5 and 12 (grey): there 0 has two supporters and is neither committed nor skipped, yet 4 lies in the history of 6, which 9, 10, 11 reference, so 6 commits. When 5 arrives, 0 passes Quorum, and its slot precedes the slot of 6 already output.